1. アーキテクチャ基盤技術

5

チップ・マルチプロセッサ

安藤 秀樹
名古屋大学
ando@cse.nagoya-u.ac.jp

ユニプロセッサからマルチプロセッサへ

プロセッサの性能は，ムーアの法則に従って長年にわたって指数関数的に向上してきた．これは，LSI 製造および設計技術，アーキテクチャ技術，コンパイラ技術など多くの技術の結集の結果である．しかし，近年，この夢のような世界も限界が見えてきた．以下にその限界要因を示す．

■ユニプロセッサの限界

LSI 製造技術：通常，プロセッサの回路は MOS トランジスタで構成されている．MOS トランジスタには，スケーリング則と呼ばれるすばらしい性質がある．これは，加工寸法をスケーリングするとさまざまな性質が良い方向にスケーリングされるというものである．ゲート遅延に関して言えば，加工寸法を $1/S$（S 倍のスケーリング）にすると，遅延も $1/S$ になる．これがプロセッサの性能の指数関数的改善の主要因である．しかし，多くの研究がなされてはいるものの，さまざまな理由で加工寸法のスケーリングの年率は鈍化の方向にある．この結果，プロセッサのクロック周波数の向上も鈍化する．

消費電力：プロセッサの消費電力は，クロック周波数の向上に伴い増加してきた．この結果，現在では安価な冷却技術では解決が困難なレベルにまで発熱するようになってきた．このような状況下では，クロック周波数を大幅に増加させることは電力制約上難しい．

命令レベル並列：一般に，プログラムの実行時間は，以下の式により与えられる：

CPU 時間＝実行命令数 /（クロック周波数× IPC）

IPC（instructions per clock cycle）は，1 クロック・サイクル当たりに処理される命令数であり，プロセッサがどの程度命令レベル並列性（ILP: instruction-level parallelism）を利用したかを示す指標である．IPC は単純には，プロセッサ内の命令ウィンドウやレジスタ・ファイルなどの資源を増加させれば向上していく．その上限はプログラムに内在する本質的な並列度によって抑えられる．この上限を調査した研究がいくつかある（たとえば文献 6）．これらの研究結果によれば，現在のプロセッサが実現している IPC はすでに上限に近く，今後資源をさらに投入しても，IPC 増加の見返りは少ない．

配線遅延：LSI 製造技術の進歩によって，ゲート遅延は減少すると述べたが，信号遅延はゲート遅延だけではない．ゲート間を接続する配線遅延が存在する．以前のプロセス世代では，配線遅延はゲート遅延とともにスケーリングされてきたが，最近では，配線遅延のスケーリングは悪くなってなっており，今後この傾向は悪化していく．これは，配線のフリンジ容量や配線間容量のスケーリングが悪いからである．一般に複雑な回路ほど配線が長くなる傾向がある．性能向上のために IPC を増加させようとすると，命令発行幅を増加させ，命令ウィンドウやレジスタ・ファイルを大きくしなければならず，回路は複雑化する．これは信号の配線遅延を増加させ，クロック周波数に悪影響を与える．つまり，IPC とクロック周波数はトレードオフの関係にあり，性能向上を制限している．

設計コストの増大：現在でもプロセッサの設計には数千人月という膨大なコストを要する．EDA ツールの進化は設計コストを削減してきたが，その速度は設計の複雑度の増加より緩やかであり，今後も設計コ

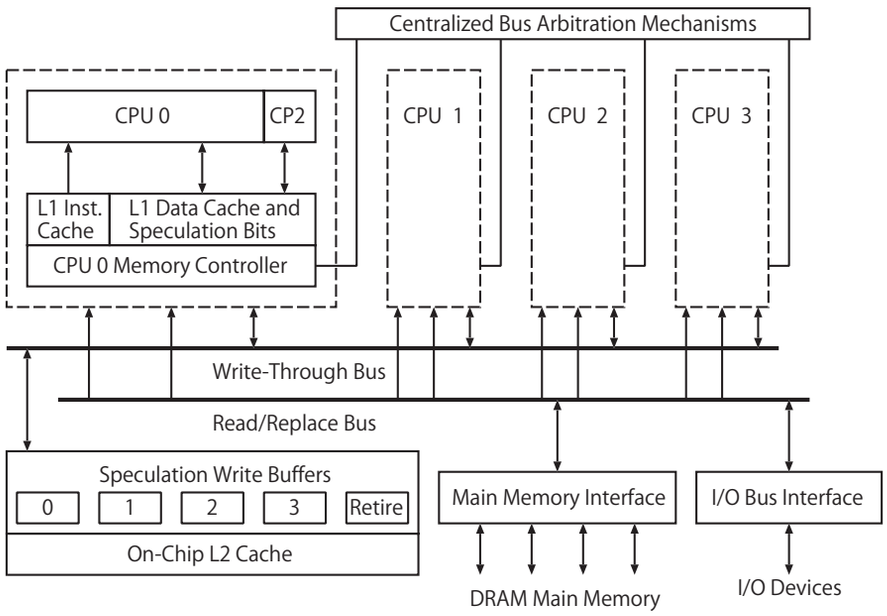

図-1　Hydraの構成
（L. Hammond, et al.: The Stanford Hydra CMP, IEEE Micro, Vol. 20, Issue 2, Mar. 2000, p.72のFigure 1を元に著者が手を加え作成した）

ストは増大していく．これに対して，コスト増に見合う性能改善の見返りは少なく，より複雑なプロセッサを設計しようとする力は弱まっている．

■チップ・マルチプロセッサの出現

前節で述べたようにユニプロセッサの性能改善には多くの難題があり，アーキテクチャ的には別のアプローチを考える必要に迫られた．その1つの答えがチップ・マルチプロセッサ（CMP: chip multiprocessor）である．CMPとは，単一チップに集積されたマルチプロセッサのことをいう．CMPが含む各ユニプロセッサのことをプロセッサ・コア，あるいは単にコアと呼ぶ．この用語を使ってCMPは，マルチコア（multi-core）とも呼ばれる．

CMPでは，コアの処理能力を強く追求しないことによりコア自身の回路規模の増大・複雑化やクロック周波数向上を抑制し，前節で示した問題を甘受あるいは回避する．その代わりとして，スレッド・レベルの並列性（TLP: thread-level parallelism）を利用することにより性能を改善させることをねらっている．

最初に商用化された汎用CMPは，IBM Power4である．今年，IntelやAMDなどからも同様のCMPが出荷された．これらは従来のユニプロセッサをベースに，単一チップに複数個ほぼ単純に集積したものである．主として従来のマルチプロセッサ応用，すなわち，独立なスレッドの実行や容易に並列化可能な応用をターゲットとしている．特に，マルチプログラミング環境で使われるサーバ用途としては自然な発展である．

これら商用の汎用CMPは，従来からあるマルチプロセッサを低コストで実現したと言えるが，同一チップ上に複数のプロセッサ・コアが集積されている利点をあまり積極的に利用するものではない．また，応用としても従来のマルチプロセッサ向けのものに力を発揮するだけであり，容易に並列化できない応用には無力である．現在のアーキテクチャ研究の舞台では，コア間をより密に結合し単一チップ集積の利点を活かすとともに，並列化困難な応用にも対応できるよう研究がなされている．次の章ではそれらを紹介する．

CMPの研究・開発事例

■ Hydra

Hydraは，スタンフォード大学で研究されているCMPである[1)]．Hydraは，後述するCMPと異なり，従来のオフチップのマルチプロセッサのアーキテクチャを踏襲した比較的保守的なアーキテクチャを採っている．**図-1**に構成を示す．同図に示すように，L1キャッシュ分散，L2キャッシュ共有のマルチプロセッサである．

Hydraの特徴は，メモリ依存に関して投機的スレッド実行を支援する機構を持っている点である．このような機構のない従来のマルチプロセッサにおけるコンパイラは，静的にデータ依存が判別できない依存に関しては，「控えめな仮定」を置かなければならなかった．こ

図-2 Hydraにおける投機的スレッド実行のためのデータ・キャッシュ・ラインのフラグ
（文献1）のp.64のFigure 7を元に著者が手を加え作成した）

こで控えめな仮定とは，依存の有無が不明な場合，依存があると考えることである．これはプログラムの意味を維持するために必要であるが，依存関係が不明なメモリ参照が多数存在するプログラムでは並列化を困難にする．実際，静的に完全にデータ依存が見える応用は配列操作を中心とするようなプログラムに限られており，ポインタを多用するプログラムでは依存関係の判定は非常に難しい．このことは，CMP の応用範囲を大きく制限している．

投機的実行を支援する機構により，Hydra では，メモリ依存の存在が曖昧な場合において，それが存在しないと仮定してスレッドを投機的に実行することができる．投機的実行結果は，その仮定が正しいと確認されたときのみ逐次状態（逐次にプログラムを実行したときのメモリ状態）を更新し，誤りの場合には安全に破棄される．これにより，コンパイラまたはプログラマは，厳密に控えめな仮定を置かなければならないという制約から解放され，プログラム内の TLP を積極的に利用することができる．

Hydra は，メモリ・システムにいくつかの変更を加え，投機的実行を支援している．その 1 つが，L2 キャッシュの前に置かれた書き込みバッファである．Hydra では，逐次状態は L2 キャッシュが保持するが，投機状態（最新のメモリ状態）は L1 キャッシュと書き込みバッファが保持する．L1 キャッシュはライトスルーであり，プロセッサからの書き込みデータは L1 キャッシュを更新するとともに，書き込みバッファに書き込まれる．書き込みバッファは複数あり，各プロセッサにはその 1 つが割り当てられる．実行が終了していない先行スレッドが存在するスレッドの実行は投機的であり，対応するプロセッサの書き込みバッファの内容は投機状態である．すべての先行スレッドが終了したスレッドの書き込みバッファのみが逐次状態となり，L2 キャッシュを更新できる．なお，スレッドの実行が終了した空きプロセッサにすぐに新たなスレッドが割り当てられるよう，書き込みバッファはプロセッサ数より 1 つ多く用意されている．

スレッド間のデータ転送を正しく行うため，あるスレッドが書き込みを行った際，後続するすべてのスレッドの L1 キャッシュの同一ラインを無効化する．当該データを読み出そうとする後続スレッドは，L1 キャッシュ・ミスを生じ，書き込みバッファまたは L2 キャッシュからデータを得る．この際，より投機的な書き込みバッファから逐次的な書き込みバッファに向かって検索を行い，データが見つかれば，そこからデータを得る．なければ，L2 キャッシュより得る．

投機的スレッド実行を支援するため，データ・キャッシュに**図 -2** に示すようなフラグを保持している．まず，メモリ依存違反（read-after-write ハザード）を検出するため，L1 キャッシュのワードごとに written ビットと read ビットと呼ぶ 2 つのフラグを用意している．written ビットは，書き込みが行われたらセットされる．read ビットは，投機スレッドが読み出しを行った場合，written ビットがセットされてなければセットされ，他のスレッドが生成したデータを読み出したことを示す．後に，read ビットがセットされたワードを先行スレッドが無効化した場合，メモリ依存違反が検出され，投機が失敗したことが知らされる．

新しくスレッドの実行が開始される際の L1 キャッシュを最新状態に変更するために，ラインごとに pre-invalidate ビットと呼ぶフラグを用意する．前述したように，書き込みは後続スレッドの同一ラインを無効化する．しかし，先行するスレッドのキャッシュは無効化されないから，それをそのまま新たなスレッドに割り当てると，キャッシュの内容が不正である．そこで，無効化信号を受けた先行スレッドは，当該ラインを無効化しないが，代わりに pre-invalidate ビットをセットする．そして，スレッドの実行が終了するとき，このフラグがセットされているラインを一斉に無効化する．こうすることにより，このプロセッサ・コアに新しくスレッドが割り当てられたとき，実行開始時の L1 キャッシュを最新状態にすることができる．

投機失敗時に投機状態を破棄するために，L1 キャッシュのラインごとに modified ビットと呼ぶフラグを用意する．このフラグは，投機スレッドによって書き込みが行われるか，書き込みバッファから投機的データを読み出したときにセットされる．投機失敗時には，modified ビットがセットされているラインが一斉に無効化され，投機状態が破棄される．

以上のように，Hydra は，メモリ依存に関するスレッドの投機的実行を支援する機構を持っており，従来曖昧なメモリ依存により並列化が困難であったプログラムにおいても，高い性能を発揮できるよう工夫されている．

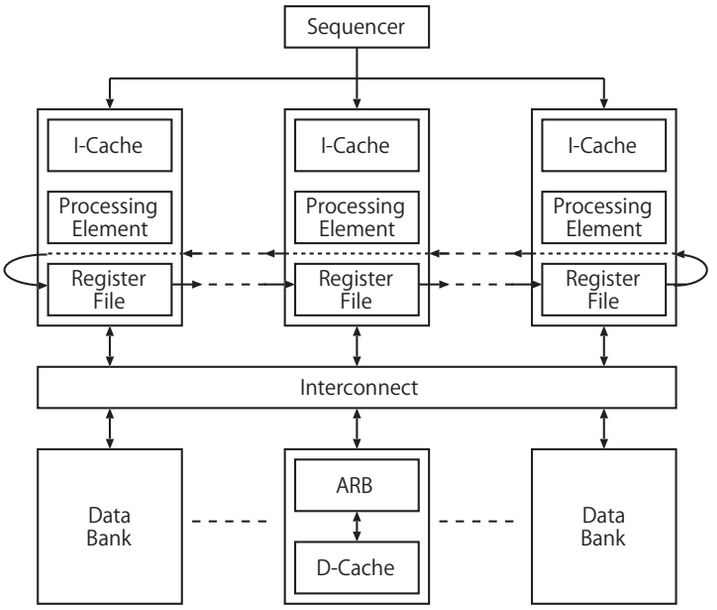

図-3　Multiscalarの構成
（文献5）のp.415のFigure 2を元に著者が手を加え作成した）

Multiscalar

Multiscalar は，ウイスコンシン大学で研究されている CMP である[5]．**図 -3** に構成を示す．プロセッサ・コアは，前節の Hydra より密に結合され，単一チップに複数のコアを集積する利点を活かしている．具体的には，コアのレジスタ・ファイルを単方向リング・バスで結合し，レジスタ・レベルでの通信を実現している．また，メモリ・システムとしては，L1 データ・キャッシュ・レベルで共有化しており，ARB（address resolution buffer）と呼ばれるバッファによってメモリ依存に関する投機的実行を支援している．

Multiscalar では，スレッドのことをタスクと呼んでいる．タスク間の通信はレジスタ・レベルで可能なため，レイテンシが小さい．この利点を活かし，コンパイラはプログラムをかなり細い粒度のタスクに分割する．各タスクは，たかだか 1 つのタスクを新たに生成する動作を逐次に行い，複数のタスクの実行をオーバラップさせることにより性能向上を図る．あるコアに割り当てられたタスクが生成するタスクは，リング・バスの方向において後続するコアに割り当てられる．タスクの生成は制御に関して投機的である．コンパイラは，タスク分割の際に，各タスクについて後続のタスク候補を列挙する．この情報は，タスク記述子と呼ばれるプログラムの属性として与えられる．タスク生成時にこのタスク記述子は，図 -3 に示すシーケンサにロードされ，実行時に次に生成すべきタスクが予測によって選択され，生成される．

Multiscalar の最大の特徴はレジスタ・ファイル間の通信機構にある．**図 -4** にその構成を示す．レジスタ・ファイルは，2 つのデータ・バンク，リング・バスに結合され書き込みを待ち合わせるキュー，各レジスタに対応する制御用のビット・マスクを持つ．まず，2 つのバンクは，投機的実行結果のコミットあるいは無効化を容易に行うためにある．2 つのバンクの対応する 2 つのレ

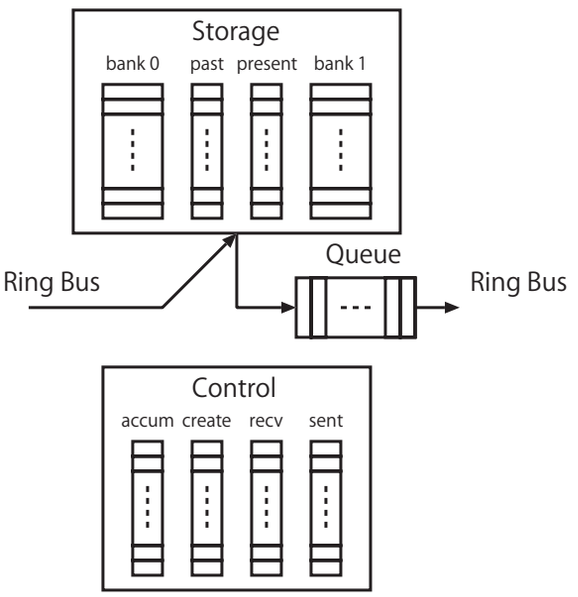

図-4　Multiscalarにおけるレジスタ・ファイル間通信機構の構成
（S. E. Breach et al.: The Anatomy of the Register File in a Multiscalar Processor, In Proc. 27th Int. Symp. on Microarchitecture, November 1994, p.184の Figure 3を元に著者が手を加え作成した）

ジスタは，論理レジスタの1つに対応している．これらを適切に操作するために，pastマスクとpresentマスクというビット・マスクを持つ．pastマスクが指すレジスタは，先行するタスクの最後の値を保持し，presentマスクが指すレジスタは，現タスクの最新の値を保持する．まず，タスク生成時にはpastマスクとpresentマスクは同一の値である（これは当該コアに以前に割り当てられていたタスクの最後のマスク状態がこのようであるためである）．先行するコアから送られてきたレジスタ値は，pastマスクが指すレジスタに書かれる．タスクの実行結果は，pastマスクが指すレジスタとは逆のレジスタに書かれ，書き込みを行ったレジスタをpresentマスクが指すようビットを更新する．命令のレジスタ参照においては，presentマスクが指すレジスタを参照する．タスクのコミットにおいては，presentマスクをpastマスクにコピーし，無効化においては，pastマスクをpresentマスクにコピーする．

後続のタスクに通信すべきレジスタは，createマスクと呼ばれるビット・マスクによって指定される．このマスクはコンパイラによって生成され，タスク記述子の一部として保存される．ただし，createマスクで指定されたレジスタを実際にプログラムのどの地点で送信するかは，forwardビットと呼ばれる命令へのタグとrelease命令と呼ばれる専用命令によって指定される．forwardビットは，後続のタスクに到達するレジスタの最後の定義をマークするもので，コンパイラによって作成される．これは，タグが付加されたopコードを新たに定義するか，または，タスク記述子と同様プログラムとは別途保存し，キャッシュに命令をロードする際に命令に付加するかのどちらかで実現するとしている．一方，後続のタスクに到達するレジスタが最後の定義以後にしか定まらない場合で，到達が決定した地点で送信を指示するものがrelease命令である．これはプログラムに埋め込まれる．forwardビットまたはrelease命令で指定されたレジスタは，レジスタ・ファイルから読み出され，リング・バスに出力される．これはキューを介して，後続のコアのpastマスクが指すレジスタに書き込まれる．レジスタを送信すると，sentマスクと呼ばれるビット・マスクの対応するビットがセットされる．レジスタを受信すると，recvマスクと呼ばれるビット・マスクの対応するビットがセットされる．

レジスタ同期を実現するために，accumマスクと呼ばれるビット・マスクが用意されている．これは受信すべきレジスタを示す．accumマスクはタスク起動時に，起動するタスクのaccumマスクとcreateマスクのビットごとの論理和により生成される．accumマスクの各ビットは，どのコアのタスクのcreateマスクによってセットされたかでタグ付けされている．これを使って，あるコアでスレッドが生成されたとき，そのコアに以前に割り当てられていたタスクによってセットされたaccumマスクのビットはクリアされる．accumマスクのビットがセットされており，対応するrecvマスクのビットがセットされていないレジスタを参照する命令の実行はストールする．accumマスクのビットがセットされていないか，セットされていても対応するrecvマスクのビットがセットされているレジスタを参照する場合は，レジスタ・ファイルよりレジスタ値を得ることができ，ストールしない．

Multiscalarはレジスタ通信機構のほかに，メモリ依存投機に関しても特徴を有している．Multiscalarでは，タスクは，メモリ依存に関して楽観的に，すなわち依存は存在しないとして投機的に実行される．依存違反はARBによって検出される．ARBは，メモリ・アドレスをインデクスとするバッファであり，各エントリは各コアに対応するフィールドを持つ．各フィールドは，対応するコアが当該アドレスに対しロードまたはストアを行ったかのビット（それぞれ，LまたはSビット），および，ストア・データを保持する領域を持つ．ロード時には，ARBを参照し，ミスならば先行するストアはないので，データ・キャッシュよりデータを得る．ヒットすれば，Sビットがセットされているデータのうち，最も最近のものが転送される．同時にLビットをセットする．ストア時には，ARBにストア・データを格納し，Sビットをセットする．後続のタスクに対応するLビットを参照し，ロードが行われているかどうかチェックする．行われていた場合，メモリ依存違反により投機失敗として無効化する．ストアを実行したタスクが実行中のタスクの中で先頭になった場合，ストア・データはデータ・キャッシュにコミットされる．

以上のように，Multiscalarは，レジスタ・レベルの通信を実現することにより，通信コストを低減し，非常に細かい粒度でのプログラムの並列化を可能にしている．また，Hydraと同様，メモリ依存に関するスレッドの投機的実行を支援する機構を持ち，並列化が困難な応用にも対応できる能力を持っている．

■ Merlot

Merlotは，NECの携帯情報端末向けCMPである[4]．**図-5**に構成を示す．4つのコアを持ち，各コアはインオーダ2命令発行のスーパースカラ・プロセッサである．A, B2つのパイプラインを持ち，両パイプラインは2つのSIMD構成を持つ32ビットのALUを持つ．Aパイプはさらに除算および積和演算が可能である．

Merlotは，0.15Mm CMOS，5メタル配線で製造され，

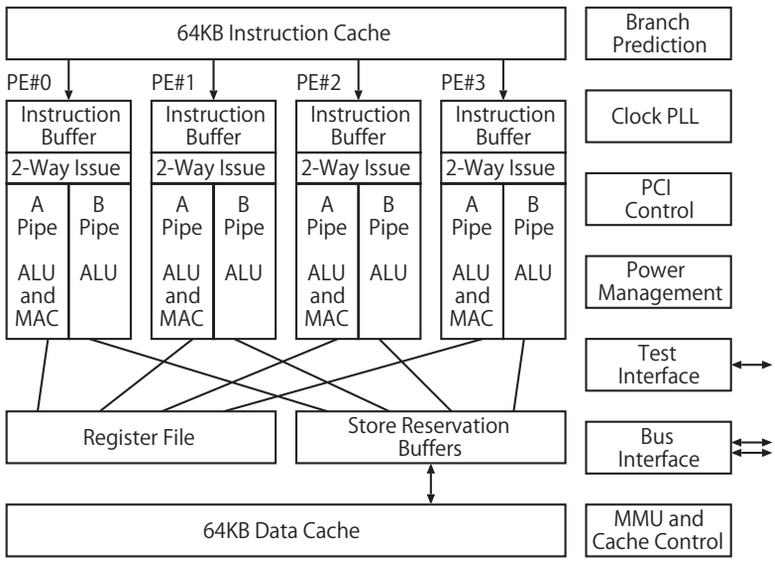

図-5　Merlotの構成
（P. N. Glaskowsky: NEC Decants Merlot, Microprocessor Report, March 20, 2000, p.1のFigure 1を元に著者が手を加え作成した）

14M のトランジスタを 10.5 × 10.5mm^2 のダイに集積している．125MHz で動作し，電源電圧 1.3V 時の消費電力が 1W と非常に低い消費電力を達成している．これは，並列処理により，目標性能達成のための動作周波数と電源電圧を低くできた結果である．

Merlot は，前述した Multiscalar と同様のスレッド実行モデルを採用している．つまり，各スレッドは新たに 1 つのスレッドを逐次に生成することによってスレッド並列実行を実現している．彼らはこれをフォーク 1 回モデル（FOPE: fork-once parallel execution）と呼んでいる．

256 × 32 ビット，16 読み出し 8 書き込みのポートを持つレジスタ・ファイルは 4 つのコア間で共有されている．ただし，各々 32 本のレジスタを持つ 8 つのセットに分離され，2 セットが 1 つのコアに割り当てられる．論理レジスタと物理レジスタの対応は，各コアが持つレジスタ・マップ表と呼ばれる表が保持する．2 つのセットの一方はフォーク前の値を，もう一方はフォーク後の実行結果を保持する．フォーク時に親スレッドの全レジスタ値は，子スレッドのレジスタの初期値として継承される．この動作はコアのマップ表をコピーすることにより行われる．これは 1 サイクルで行われ，スレッドの初期値の通信コストを最小化している．

Merlot は 2 つのタイプのスレッドの投機的実行を支援している．1 つは分岐に対する投機（彼らは制御投機と呼んでいる）であり，もう 1 つはメモリ依存に対する投機（彼らはデータ投機と呼んでいる）である．制御投機は，たとえば，繰り返し回数が動的にしか分からないループのイタレーションを並列に実行するのに効果的である．データ投機の有効性は，Hydra の節で述べた通りである．投機スレッドのストア・データは，データ・キャッシュの前方に置かれたストア・リザベーション・バッファにいったん蓄えられ，投機成功時にデータ・キャッシュにコミットされ，失敗時にフラッシュされる．発表時の実装では，ストア・リザベーション・バッファのエントリ数は 6 である．

いずれの投機も，スレッドの生成時に命令によって明に指示される．制御投機においては，投機スレッドのコミットおよび破棄も，親スレッドに命令を挿入することで指示される．データ投機では，依存違反はストア・リザベーション・バッファによって検出される．データ投機されたスレッドの実行結果は，依存違反なく実行が終了し，かつ，親スレッドの実行が終了した際にコミットされる．投機失敗の際には，スレッドは再実行される．

NEC によれば，Merlot は汎用ではないが，積極的な投機支援と低レイテンシ通信機構を有し，ほとんど汎用 CMP といってよい構成となっている．このような CMP を実際に設計し，並列化により非常に小さな電力しか要しないことを実証した点で高く評価される．

今後の展開

CMPの設計では種々のトレードオフが存在する．その1つが，コアの複雑度とコア数である．コアの複雑度を増加させ単一スレッドのILPを多く利用できるようにすれば，同一面積のダイに集積できるコアの数が減少しTLPを多く利用できない．逆に，コアを単純にすると，多くのコアを集積できTLPを多く利用できるが，単一スレッドの性能は犠牲になる．

このトレードオフに関連するが，集積するコアの均質性の問題，すなわち，集積するコアを同種（homogeneous）にするか異種（heterogeneous）にするかという問題がある．組み込み用途では，異種構造がよく採用されている．1つまたは複数の同種汎用コアのほかに計算を助ける異種コアを集積する．このようなCMPは，各コアが応用の性能要求によく適合した場合に，高いコスト性能比を得ることができる．組み込み用途ではこれが可能である．しかし，どのような応用が実行されるか予測困難な汎用では，適切な解とは言えない．

異種CMPにおける問題を部分的に緩和する解として，共通の命令セット・アーキテクチャ（ISA: instruction set architecture）を持ち，複雑度を異にするコアを集積する異種CMPが考えられる．このようなCMPでは，汎用においても，応用とコアの適合性の問題が前述の異種CMPに比べ小さい．ISAが共通であるから，どのような局面においても応用をどのコアにも割り当てることができるからである．このようなCMPは，応用の持つ並列性の多様性に対してうまく対応できるという利点を持っている．すなわち，複雑度の低いコアを多数集積することによりTLPが豊富な応用や局面にうまく対応できる一方で，複雑度の高いコアも同時に持ち合わせることによりTLPに乏しくILPに頼らざるを得ない応用にも対応できる．この結果，単純な同種CMPより高いコスト性能比を得る可能性を持っている[2]．

以上のようなトレードオフが考えられるが，今のところ，多くの商用汎用CMPは，ILP利用を重視した複雑な同種コアを少数集積する解を採っている．おそらくこれは，1）異種CMPにおける応用とコアの適合の問題が嫌われている，2）メーカ間の厳しい競争上，すぐに対応可能な当面の解が選ばれていることが理由と思われるが，今後の研究と投資によって，各メーカが考える最適解は変わっていく可能性は十分にある．

最初に述べたように，現在の汎用CMPは，従来のマルチプロセッサ応用をターゲットとしている．今後，ユニプロセッサ応用の性能が重要なPCのプロセッサとして，CMPが有効な解となるには，さらなる研究が必要である．これに関して最も本質的な疑問は，ユニプロセッサ応用にはどれほどのTLPが存在し，その量はCMPを活かせるほど十分かというものである．これへの答えとして，たとえば，文献3）で整数系プログラムにどの程度のTLPが存在するかが調査されている．この調査によれば，基本ブロック・レベルでの制御依存解析とそれに基づくスレッド分割および投機的実行を行えば，引き出せるTLPの量はSPECint95で5～20程度はあり，ユニプロセッサ応用においてもCMPを十分活かせるとしている．

ユニプロセッサ応用に内在するTLPを引き出し性能に結びつけるには，少なくともハードウェアとしては，本稿で挙げた研究でなされているような投機や細粒度スレッドへの支援が重要である．さらには，そういった支援を最大限活用し，十分な並列性を持ったスレッドにプログラムを分割するコンパイラの開発も重要である．今後，一層の研究が必要とされている．

参考文献

1）Hammond, L., Willey, M. and Olukotun, K.: Data Speculation Support for a Chip Multiprocessor, In Proc. Eighth Int. Conf. Architectural Support for Programming Languages and Operating Systems, pp.58-69 (Oct. 1998).

2）Kumar, R., Tullsen, D. M., Ranganathan, P., Jouppi, N. P. and Farkas, K. I.: Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance, In Proc.31st. Int. Symp. on Computer Architecture, pp.64-75 (June 2004).

3）Nakajima, A., Kobayashi, R., Ando, H. and Shimada, T.: Limit of Thread-Level Parallelism on Partitioning Levels and Speculations in Non-Numerical Programs, In Proc. Eighth International Symposium on Low-Power and High-Speed Chips, pp.465-472 (Apr. 2005).

4）Nishi, N., Inoue, T., Nomura, M., Matsushita, S., Torii, S., Shibayama, A., Sakai, J., Ohsawa, T., Nakamura, Y., Shimada, S., Ito, Y., Edahiro, M., Mizuno, M., Minami, K., Matsuo, O., Inoue, H., Manabe, T., Yamazaki, T., Nakazawa, Y., Hirota, Y., Yamada, Y., Onoda, N., Kobinata, H., Ikeda, M., Kazama, K., Ono, A., Horiuchi, T., Motomura, M., Yamashina, M. and Fukuma, M.: A 1GIPS 1W Single-Chip Tightly-Coupled Four-Way Multiprocessor with Architecture Support for Multiple Control Flow Execution, 2000 IEEE Int. Solid-State Circuits Conference Digest of Technical Papers, pp.418-419 (Feb. 2000).

5）Sohi, G. S., Breach, S. E. and Vijaykumar, T. N.: Multiscalar Processors, In Proc. 22nd. Int. Symp. on Computer Architecture, pp.414-425 (June 1995).

6）Wall, D. W.: Limits of Instruction-Level Parallelism, In Proc. Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pp. 272-282 (Apr. 1991).

（平成17年8月18日受付）